

Building a Simple Application

Any kind of practical security is an ongoing process. You don't just set up security one day and then never worry about it again. Even the best application programming methods won't replace maintenance like performing security audits and monitoring unusual user-access patterns. To give you a good example of how important a complete security effort is, consider a recent Caterpillar, Inc. break-in by relatively novice crackers. It seems that the system administrator had left a default password in place, which allowed the crackers to bypass security with ease. In other words, the system administrator left the door wide open for the crackers.

Normally, programmers don't have to worry about security on a daily basis as a system administrator does. You'll write code that's, hopefully airtight and relies on the security API provided with Windows 2000. However, it's important to realize that the code you write is only part of the picture. It's extremely important for everyone involved in maintaining system security to understand how the operating system maintains a secure environment. Once you know how things work, you can write code that better enables the application to maintain a secure environment.

It's up to the programmer to help the system administrator with the human side of the security picture as well. You can include application code that checks the security measures that a user is using and ensures that those measures comply with the settings provided by the system administrator. In addition, you can keep the system administrators from shooting themselves in the foot. For example, you can include code that automatically asks the system administrator if he or she wants to delete an account that hasn't been used for a given amount of time. In addition, your application can force

In This Chapter

Understanding how Windows 2000 security works

How security technology interoperability concerns affect the programmer

Working with access tokens and security descriptors

Addressing object-level security issues

Understanding Kerberos, public-key infrastructure, and DPA support

Working with Windows 2000 data-encryption support

Keeping your Internet transaction secure

How to create a secure DCOM environment

Understanding DCOM access security, launch security, identity, and connection policy issues

Working with the Security Configuration Editor

the user to use good security habits like requiring difficult-to-guess passwords of sufficient length to make it hard for a cracker to use computing power to overcome security as well.

Note

This chapter provides you with an overview of the theory behind Windows 2000 security. We'll look at potential solutions for generic problems. What you shouldn't be looking for in this chapter are concrete answers that are going to work in your specific situation. It would be silly to try to cover all of the various technologies available today, or even every security issue you'll ever run into, in one chapter. What we'll concentrate on instead is the programmer's perspective on security matters. We'll talk about the technologies available to you as a programmer, but we won't get into specific solutions to a specific problem. The solution you finally decide to use will have to be based on your company's needs and the tools you have at your disposal.

The first thing we'll look at is Windows 2000 security as a whole. Then we'll look at the basis for object-level security in Windows 2000. If you already know how access tokens and security descriptors work, you can potentially skip this section of the chapter. We'll look at how these architectural elements work at a very simple level.

The next section of the chapter is a must-read for anyone who wants to learn about the latest security features that Windows 2000 has to offer. We'll look at the new Kerberos and public-key infrastructure (PKI) additions to Windows 2000. These new technologies not only make your network more secure, they do it with less stress on the part of both the administrator and developer.

Note

For the purposes of this chapter, the term *cracker* will always refer to an individual that's breaking into a system unauthorized. This includes any form of illegal activity on the system. On the other hand, a *hacker* will refer to someone who performs low-level system activities, including testing system security. Sometimes you need to employ the services of a good hacker to test the security measures you have in place, or suffer the consequences of a break-in. This chapter will use the term *hacker* to refer to someone who performs these legal forms of service.

By the way, this is the way hacker and cracker are commonly defined in security books and by most security professionals. The trade press usually uses the terms incorrectly.

We'll discuss data-encryption concerns will appear in the next section of the chapter. It's not enough to make sure that your system is secure if the data being transmitted from site to site is open for scrutiny. Today's computing environment requires security on many levels including both system-level and data-level security.

Normally, data gets transmitted from one application to another in a fairly straightforward manner. There are some special concerns when it comes to security object-level communication, especially the kind promoted by DCOM, because the data is no longer being transmitted in the traditional way. Using DCOM allows you to create multitiered applications that can span several servers, but this flexibility also requires a special awareness of security concerns on the part of the programmer.

The final section of the chapter looks at a fairly recent addition to Microsoft's arsenal of utilities — the Security Configuration Editor. As a programmer, you need to know how to set up various security scenarios while testing your application. What this means is that you should be familiar with every administrator-level tool, as well as the special programming tools at your disposal. The Security Configuration Editor is an important new addition to the war on computer crime.

An Overview of Windows 2000 Security

A famous cracker once said that it's not a matter of if, but when, someone will break into your system. This statement can easily be extended to any computer scenario. It's important to realize that a determined individual can always break your security, so constant hardware/software upgrades and scrutiny on the part of both developer and network administrator are essential. For example, the simple act of tracking security events will allow you to detect security failures before they become a problem.

Windows 2000 provides a level of security for an operating system that almost verges on paranoia. It allows you to set security in a variety of ways, including both the familiar user and file levels. You can also create groups and assign security by using groups instead of individuals. In addition, you can monitor every aspect of the security system using various alarms and log files. Windows 2000 excels in the way that it actually monitors system activity. Not one event goes without some kind of scrutiny. In fact, the simple act of passing information from one process to another undergoes some level of scrutiny by Windows 2000.



Smaller companies can now take advantage of an alternative to Microsoft's offerings. RSA recently introduced the Keon Software Suite of products. This suite includes a security server, desktop-component, agents, SDK, and a certificate server (most likely tied in with VeriSign in some way). Keon is designed to help companies get past some of the interoperability problems they've experienced in the past by offering a complete security solution. The SDK will allow programmers to develop add-on products for the suite, making it possible to tie the suite into custom applications. You can find out more about Keon at <http://www.rsa.com>.

Developers need to address two main levels of security: local and remote. A user logging onto a company terminal is an example of local security. This is the kind of security that developers have had to deal with for quite some time, and the procedures for doing so are fairly well known. Local access also implies fewer potential security problems because you can secure the lines of communication between the client and server. Less well-known are the procedures for remote security. For example, how does a server-side component deal with a user who is attempting access from an Internet connection rather than a local connection? The data flow between the component and client is now a source of concern — something that the developer will need to address as part of the component.

Fortunately, the Windows 2000 Security API can help you create a more secure local, intranet, and Internet environment. Adding API calls to your application, especially for new features, can make local access much more secure. Obviously, many of these new features were designed with the Internet in mind. Fortunately, you can mix these new solutions with older technologies. You can use ActiveX controls to access Windows 2000 security features from both the server and the client site, ISAPI extensions to access them from the server side, and ISAPI filters to perform behind-the-scenes monitoring of security events.

Whether you're creating an application for Internet, intranet, local, WAN, or LAN use, you'll find that a good understanding of the underlying network security architecture is essential. In this section, the term *application* is used to refer to any number of executable formats, including standard applications like a word processor, ActiveX controls, and standard DLLs. As previously stated, Windows 2000 provides a wealth of security features that you can use to make your information more secure. However, we can break the low-level implementation details of those technologies down to make them both more generic and easier to understand. That's the purpose of this section—to help you understand the low-level methods that Microsoft uses to make security work in Windows 2000.

Windows 2000 and Windows 95/98 both use the term *object* rather loosely. It's true that a lot of objects are lurking beneath the surface, but you may find that they don't fit precisely within the C++ usage of the term. In fact, in many cases, these Windows objects don't necessarily fit within the COM definition of an object either. In the next few sections, we'll look at an object as the encapsulation of code and data required to perform a specific security task. In other words, each security object is a self-contained unit designed to fulfill a specific role.

Security Issues across Mixed OS Environments

A lot of developers will find themselves creating applications for mixed operating system environments that include both Windows 95 and Windows 98. Unfortunately, Windows 95/98 doesn't provide the same level of security that Windows 2000 does, so you'll find yourself doing without added security under Windows 95/98 at times. However, when Windows 95/98 does provide a security feature, it uses the same setup as Windows 2000, so one security module will work with both of them. Of course, sometimes you'll definitely want to detect the operating system and then use a separate DLL module for Windows 2000 to make better use of its enhanced security capabilities.

For those of you that are developing in a mixed operating system environment, remember that the following discussion applies equally well to Windows 2000 and Windows 95/98. The main difference is that Windows 95/98 won't support some of the features that Windows 2000 does.

**Note**

In many places in both Windows 95/98 and Windows 2000, Microsoft chose to use the full C++ definition of an object mainly because they implemented the required functionality using the MFC libraries. In other words, you may find that the full COM implementation of the object is missing, at least with regard to the definition of COM as Microsoft has spelled it out in the specification. However, when reading either the Microsoft documentation or this chapter, you shouldn't depend on the definition of an object to mean a strict C++ object—think of an object more in the generic sense of the word. (Theoretically, all parts of the operating system, including security, will eventually use COM exclusively, but that isn't the case right now.)

Knowing that everything is an object makes security a bit easier to understand—at least it's a starting point. However, objects themselves are just a starting point. Users are the other part of the security equation. An object is accessed by a user—so security in Windows is a matter of comparing the object's protection to the user's rights. If the user has sufficient rights (rights that meet or exceed those of the object), then he or she can use the object. The Windows documentation refers to an object's level of protection as a security descriptor. This is the structure that tells the security system what rights a user needs to access the object. Likewise, the user has an access token, which is another structure that tells the security system what rights a user has in a given situation. “Token” is a good word here because the user will give Windows 2000 the token in exchange for access to the object. (Think of the object as a bus, with Windows 2000 as the driver and the user presenting the required token to board.) Figure 17-1 shows both of these structures.

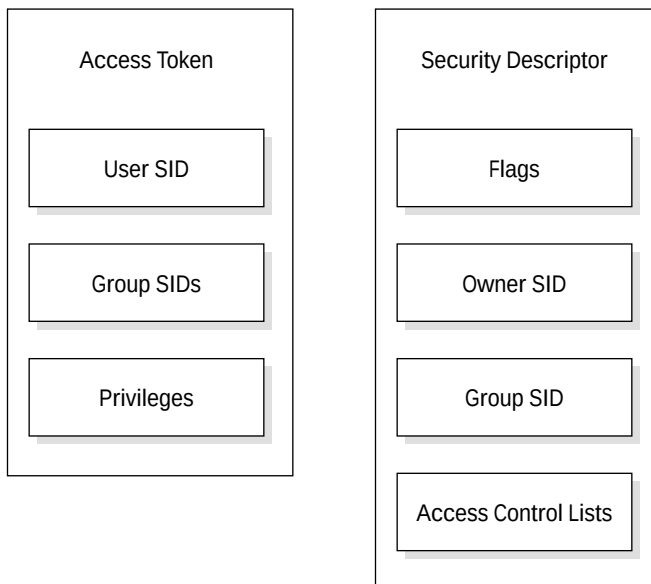


Figure 17-1: Access tokens define the user's rights, whereas security descriptors define the protection level for a process.

This is the shortest look you can take at security under either Windows 95/98 or Windows 2000. Simply knowing that there are security objects and user tokens will go a long way toward helping you make sense out of the Windows security API calls. The following sections of the chapter will address four issues. We'll look at the low-level theory behind Windows 2000 security, which means looking at access control lists (ACLs) and security descriptors. Unfortunately, things aren't always as easy as they could be when working with Microsoft's security strategy, so the next topic we'll cover is potential programming pitfalls that you need to be aware of. For example, you can't simply access the ACLs in any order; you need to use certain procedures when building a security-enhanced application of any type. We'll also look at how these low-level security features impact the functionality of your application. For example, the user will need to work with the various security screens that you'll have to add to your application. At the very least, the user will need to know what to do when they see a password dialog appear after requesting access to the company database.

Addressing Security Interoperability Concerns

Most of the security software that you can use today is vendor-specific and proprietary, which means that you're stuck looking at security through one vendor's eyes. The problem with this approach is that you may not find a single vendor who can address all of your security concerns. Sure, Windows 2000 itself is protected by the security built in by Microsoft, but today's computing environment doesn't limit itself to just one isolated company.

In a world-based economy, you need the ability to secure a wide range of computer equipment from a large number of security threats. The ability to make security solutions from more than one vendor work together is no longer a luxury—it's a necessity.

Security vendors see this problem as well and are working to make their software interoperable through a number of initiatives and APIs. The latest initiative to hit the market is Common Content Inspection (CCI) API, a project started by Aventail, Corp, Finjan Software, Ltd., and Check Point Software Technologies. This group joins others like Adaptive Network Security Alliance (ANSA) and Open Platform for Secure Enterprise Connectivity (OPSEC) Alliance. The goals of these three groups vary slightly, but the main goal is to make it possible for more than one vendor's products to work together. The largest of these groups is OPSEC Alliance with 260 members (as of this writing).

The CCI API seeks to promote interoperability through the use of common interfaces for inspecting content. In other words, each vendor's software might provide completely different features and capabilities, but they would be able to work with the same data because the means for inspecting it are the same. The resulting specification is supposed to work with data at whatever level it appears at, including firewalls, antivirus checks, and mobile code. You can find out more about CCI API at <http://www.stardust.com/cciaapi/> and <http://www.finjan.com/cciaapi.cfm>.

OPSEC Alliance is providing a similar set of APIs through Convent Vectoring Protocol (CVP). The difference, in this case, is that they're concentrating on a single firewall product, Check Point's Firewall-1. In addition to content, OPSEC is working on APIs that allow various vendor products to work together at the application level. At the time of this writing, the OPSEC Alliance offering is pretty much in place and even includes a certification program. You can find out more about OPSEC at <http://www.opsec.com/index.html>.

Like the OPSEC Alliance, ANSA caters to a specific vendor product. In this case, the target software is Internet Security System (ISS) Group's intrusion detection software. ANSA currently has 40 companies participating in the standardization effort. Unlike the other two security interoperability efforts discussed in this section, ANSA offers a software development kit (SDK) that contains Adaptive Network Security (ANS) modules designed to make interoperability easier. The ANS modules will allow ISS Group intrusion detection software to issue alerts to firewalls, extranet management applications, and other software as soon as an intrusion is detected. These other applications would then deny system access to the intruder. You can find out more about ANSA at <http://ansa.iss.net/>.

Understanding Access Tokens

You'll find that there are two of ways of looking at a user's rights under Windows—both of them related to objects in one form or another. The user's access token has a security identifier (SID) to identify the user throughout the network—it's like having an account number. The user token that the SID identifies tells which groups the user belongs to and what privileges the user has. Each group also has a SID, so the user's SID contains references to the various group SIDs that the user belongs to, not to a complete set of group-access rights. You'd normally use the Local Users and Groups snap-in (an add-in module) for Microsoft Management Console (MMC) under Windows 2000 to change the contents of this access token. (This same snap-in can be used to manage remote machines that support remote management.) Figure 17-2 shows an example of the Local Users and Groups snap-in.



Figure 17-2: The first place to start looking for the effects of access tokens is within the Local Users and Groups display.

So, what's the privileges section of the access token all about? Figure 17-3 shows a breakout of the elements within the privileges section. It begins with a count of the number of privileges that the user has—not the groups that the user belongs to, but the number of special privilege entries in the access token. This section also contains an array of privilege entries. Each privilege entry contains a locally unique identifier (LUID)—essentially a pointer to an object—and an attribute mask. The attribute mask tells what rights the user has to the object. Group SID entries are essentially the same. They contain a privilege count and an array of privilege entries.

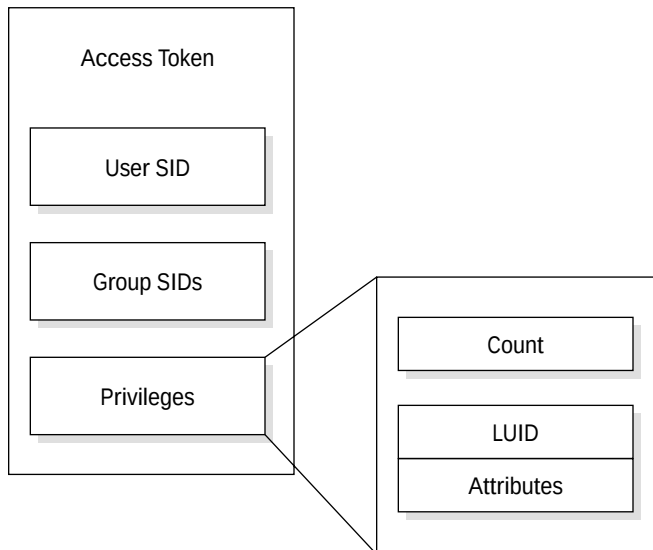


Figure 17-3: The Privileges section of the access token determines the number and type of rights the user has.

Tip

Now is probably a good time to look at the Windows API help file provided with your copy of Visual C++ to see what kind of SID- and token-related API calls you can find. Examples of SID-related calls include `CopySID` and `AllocateAndInitializeSID`. You'll also find that the `OpenProcessToken` and `GetTokenInformation` calls are essential to making security work correctly with any language you use.

One of the things that you need to know about working with some kinds of objects is that object rights flow down to the lowest possible node unless overridden by another SID. For example, if you give a user read-and-write rights to the `\Temp` directory on a hard drive, those rights are also applied to the `\Temp\Stuff` directory unless you assigned the user specific rights to that directory. The same is true for containers. Assigning a user rights to a container object such as a Word document gives the user the right to look at everything within that container, even other files in most cases. The ability of rights to flow down from an upper level of the hierarchy to a lower level means that you have to be careful when assigning rights from within an application. The rights you assign to the user at an upper

level could have unanticipated and unwanted effects at a lower level of the object hierarchy.

Now that you have a better idea of how access tokens work, let's talk about the access token calls that the Windows API provides, since they're the first stepping-stone that you'll need to know about when it comes to security. To do anything with a user's account—even if you want to find out who has access to a particular workstation—you need to know about tokens.

As previously stated, tokens are the central part of the user side of the security equation. You'll almost always begin a user account access with a call to the `OpenProcessToken` function. Notice the name of this call—it deals with any kind of a process, user or otherwise. The whole purpose of this call is to get a token handle with specific rights attached to it. For example, if you want to query the user account, you need the `TOKEN_QUERY` privilege. (Your access token must contain the rights that you request from the system, which is why an administrator can access a token, but users without administrator rights can't.) Any changes to the user's account require the `TOKEN_ADJUST_PRIVILEGES` privilege. There are quite a few of these access rights, so we won't go through them all here.

Tip

Windows 2000 does support many more Windows security API calls than Windows 95/98 does because its security is much more robust. In fact, you'll find that your ability to manage security when using Windows 95/98 is severely hampered by its lack of support for Windows 2000 security features. For example, you can't use the `GetUserObjectSecurity` call under Windows 95/98. Most of the access token calls that we'll look at in the example security applications in this book won't work either. The best way to figure out whether a call is supported or not is to test it. If you get `ERROR_CALL_NOT_IMPLEMENTED` (value 120) returned from the call, you know that you can only use it under Windows 2000.

Once you have an access token handle, you need to decide what to do with it. If you decide you want to change a user's privilege, you need the LUID for the privilege you want to change. All of these appear in the `WINNT.H` file with an `SE_` attached to them. For example, the `SE_SYSTEM_PROFILE_NAME` privilege allows the user to gather profiling information for the entire system. Some SE values aren't related to users (for example, the `SE_LOCK_MEMORY_NAME` privilege that allows a process to lock system memory). You get the LUID for a privilege using the `LookupPrivilegeValue` call. Now you can combine the information you've gotten so far to change the privilege. Most generally, you'll use the `AdjustTokenPrivileges` call to make the required change.

Tip

Look in the `WINNT.H` include file for a list of Windows 2000-defined privileges that you can change.

Querying the user's account (or other access token information) is fairly straightforward. You use the `GetTokenInformation` call to retrieve any information you might need. This call requires a token class parameter, which tells Windows what kind of information you need. For example, you'd use the `TokenUser` class if you wanted to know about a specific user. You'll also need to supply an appropriate

structure that Windows can use for storing the information you request — which differs based on the token class you request.

Understanding Security Descriptors

It's time to look at the security descriptor. Remember that the security descriptor determines which rights a user will need (through that access token) to gain access to a specific object. Figure 17-4 shows that each security descriptor actually contains five main sections, including two access control lists (ACLs). The first section is a list of flags. These flags tell you the descriptor revision number, format, and ACL status.

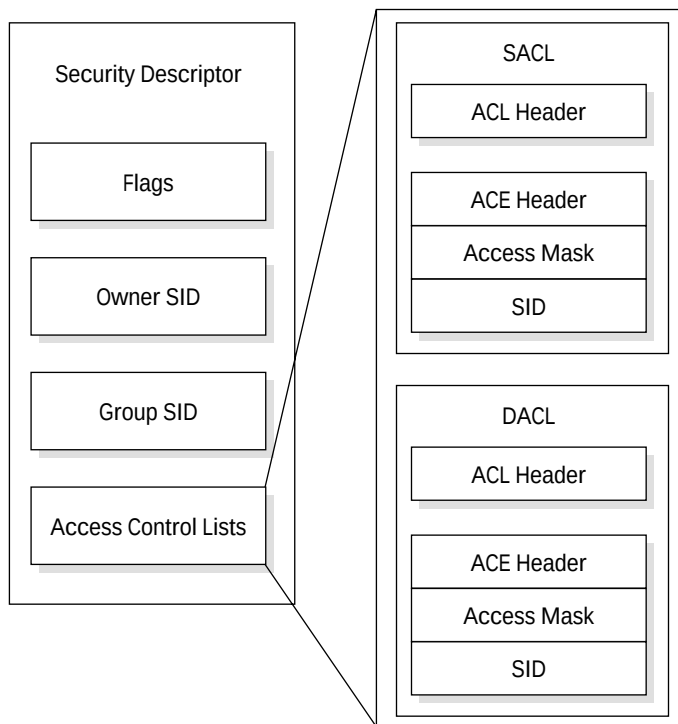


Figure 17-4: The security descriptor includes two ACLs, which determine the object's security settings.

The next two sections contain SIDs. The owner SID tells who owns the object. This doesn't have to be an individual user; Windows allows you to use a group SID here as well. The one limiting factor is that the group SID must appear in the access token of the person changing the entry. The group SID allows a group of people to own the object. Of the two SIDs, only the owner SID is important under Windows. The group SID is used as part of the Macintosh and POSIX security environment.

The final two sections contain ACLs. The security access control list (SACL) controls Windows' auditing feature. Every time a user or a group accesses an object and the auditing feature for that object is turned on, Windows makes an entry in the audit log. The discretionary access control list (DACL) controls who can actually use the object. You can assign both groups and individual users to a specific object.

**Note**

There are actually two types of security descriptors: absolute and self-relative. The absolute security descriptor contains an actual copy of each ACL within its structure. This is the type of security descriptor to use for an object that requires special handling. The self-relative security descriptor contains only a pointer to the SACL and DACL. This type of descriptor saves memory and reduces the time required to change the rights for a group of objects. You'd use it when all the objects in a particular group require the same level of security. For example, you could use this method to secure all the threads within a single application. Windows requires that you convert a self-relative security descriptor to absolute format before you can save it or transfer it to another process. Every descriptor you retrieve using an API call is of the self-relative type—you must convert it before you can save it. You can convert a security descriptor from one type to another using the `MakeAbsoluteSD` and `MakeSelfRelativeSD` API calls.

An ACL consists of two types of entries. The first entry is a header that lists the number of access control entries (ACEs) that the ACL contains. Windows uses this number as a method for determining when it has reached the end of the ACE list. (No end-of-structure record or any way of determining a precise size for each ACE in the structure exists.) The second entry is an array of ACEs.

**Caution**

Never directly manipulate the contents of an ACL or SID because Microsoft may change their structure in future versions of Windows. The Windows API provides a wealth of functions to change the contents of these structures. Always use an API call to perform any task with either structure type to reduce the impact of changes in structure on your application.

So what is an ACE? An ACE defines the object rights for a single user or group. Every ACE has a header that defines the type, size, and flags for the ACE. Next comes an access mask that defines the rights that a user or group has to the object. Finally, there's an entry for the user's or group's SID.

There are four different types of ACE headers. Only three of these ACE header types are used in the current versions of Windows. The following list describes each of the ACE header types in detail.

Access-Allowed	Appears in the DACL and grants rights to a user. You can use it to add to the rights that a user already has to an object on an instance-by-instance basis. For example, say you wanted to keep the user from changing the system time so that you could keep all the machines on the network synchronized. However, there might be one situation—such as daylight savings time—when the user would need this right. You could use an access-allowed ACE to give the user the right to change the time in this one instance.
Access-Denied	This ACE header type also appears in the DACL. It revokes rights that the user has to an object. You can use it to deny access to an object during special system events. For example, you could deny access rights to a remote terminal while you perform some type of update on it.
System Audit	Works with the SACL. It defines which events to audit for a particular user or group.
System Alarm (currently unused)	Allows either the SACL or DACL to set an alarm when specific events happen.

Tip

This is a good time to look through the Windows API help file to see what types of access rights Windows provides. You should also look at the various structures used to obtain the information. Especially important are the ACL and ACE structures. Look for the ACE flags that determine how objects in a container react. For example, check out the `CONTAINER_INHERIT_ACE` constant that allows subdirectories to inherit the protection of the parent directory.

Understanding what a security descriptor is and how the various structures it contains interact is only one part of the picture. You also need to know how to begin the process of actually accessing and using security descriptors to write a program.

The first thing you need to understand about security descriptor access is that, unlike tokens, security descriptors aren't generalized. You can't use a standard set of calls to access them. In fact, there are five classes of security descriptors, each of which uses a different set of descriptor calls to access the object initially. (You must have the `SE_SECURITY_NAME` privilege to use any of these functions.) Table 17-1 describes each of the security descriptor classes.

Table 17-1
Security Descriptor Class Descriptions

<i>Class</i>	<i>Associated Call</i>	<i>Description</i>
Files, directories, pipes, and mailslots	GetFileSecurity and SetFileSecurity	There are multiple forms of data storage in Windows. In many cases, they use separate calls because you have to interact with them in a variety of ways. In this particular case, storage of any kind uses the same file-oriented calls.
Processes, threads, access tokens, and synchronization objects	GetKernelObjectSecurity and SetKernelObjectSecurity	All of these objects, even the access tokens, are actually kernel objects. As such, they also have their own security descriptor for protection purposes.
Window stations, desktops, windows, and menus	GetUserObjectSecurity and SetUserObjectSecurity	A window station is a combination of the hardware you use to access the system — keyboard, mouse, and screen. Desktops contain windows and menus—the display elements you can see onscreen. These four objects inherit rights from each other in the order shown. In other words, a desktop will inherit the rights of the window station.
System registry keys	RegGetKeySecurity and RegSetKeySecurity	Notice that these two calls start with Reg, just like all the other registry-specific calls that Windows supports.

Continued

Table 17-1 (continued)

Class	Associated Call	Description
Executable service objects	QueryServiceObjectSecurity and SetServiceObjectSecurity	For some strange reason, neither call appears with the other security calls in the Windows API help file. You'll need to know that these calls exist before you can find them. An executable service is a background task that Windows provides—such as the UPS monitoring function. You'll find the services that your system supports by double-clicking the Services applet in the Control Panel.

 Note

Only the NTFS file system under Windows 2000 provides security. The VFAT file system (FAT32) provides it to a lesser degree under Windows 95/98. You can't assign or obtain security descriptors for either the HPFS or FAT file systems under either operating system. The FAT file system doesn't provide any extended attribute space, one requirement for adding security. The HPFS file system provides extended attributes, but they don't include any security features. Of all the file systems described, NTFS is the most secure. However, never assume that any file system is completely secure. There are utility programs on the Internet that will read the contents of an NTFS file partition even if the user hasn't logged in properly to Windows 2000.

Once you do gain access to the object, you'll find that you can perform a variety of tasks using a generic set of API calls. For example, the `GetSecurityDescriptorDACL` retrieves a copy of the DACL from any descriptor type. In other words, the descriptors for all these objects follow roughly the same format—even though the lengths of most of the components will differ. One reason for the differences in size is that each object will contain a different number of ACEs. The SIDs are different sizes as well.

The next step in the process of either querying or modifying the contents of a security descriptor is to disassemble the components. For example, you could view the individual ACEs within a DACL or a SACL by using the `GetACE` API call. You could also use the owner and group SIDs for a variety of SID-related calls (we discussed these calls in the “Understanding Access Tokens” section of the chapter).

Suffice it to say that you can use a generic set of functions to manipulate the security descriptor once you obtain a specific procedure.

In essence, any security descriptor access will always consist of the same three steps:

1. Get the descriptor.
2. Remove a specific component.
3. Modify the contents of that component.

To change the security descriptor, you reverse the process. In other words, you use a call like `AddACE` to add a new ACE to an ACL, then use `SetSecurityDescriptorSACL` to change SACL within a descriptor, and finally, save the descriptor itself using a call like `SetFileSecurity` (assuming that you want to modify a file object).

Potential Programming Pitfalls When Using Windows 2000 Security

Once you start thinking about the way Windows evaluates the ACEs in the DACL, you'll probably discover a few potential problem areas—problems that the Windows utilities take care of automatically, but which you'll need to program around in your application to derive the same result. (The SACL has the same potential problem, but it affects only auditing, so the effect is less severe from the standpoint of system security.)

Windows evaluates the ACEs in an ACL in the order in which they appear. At first this might not seem like a very big deal. However, it could become a problem in some situations. For example, what if you want to revoke all of a user's rights in one area, but his or her list of ACEs includes membership in a group that allows access to that area? If you place the access allowed ACE first in the list, the user would get access to the area—Windows stops searching the list as soon as it finds the first ACE that grants all the user's requested rights (or an ACE that denies one of the requested rights). Granted rights are cumulative. If one ACE grants the right to read a file and another the right to write to it, and the user is asking for both read and write rights, Windows will view the two ACEs as granting the requested rights. The bottom line is that you should place all of your access-denied ACEs in the list first to prevent any potential breach in security.

You also need to exercise care in the ordering of group SIDs. Rights that a user acquires from different groups that he or she belongs to are cumulative. This means a user who is part of two groups, one that has access to a file and another that doesn't, will have access to the file if the group granting the right appears first on the list.

Obviously, you can spend all of your time trying to figure out the best arrangement of groups. As the number of groups and individual rights that a user possesses increase, so does the potential for an unintended security breach. That's why it's important to create groups carefully and limit a user's individual rights.

You need to be concerned about two other things when you look at security under Windows 2000 as a cracker. The first deals with a client's ability to access data he or she isn't supposed to when accessing data through a server. (I'm not talking about a file server here, but some type of DDE or other application server.) Think about it this way: What if a client didn't have rights to a specific type of data, but accessed the data through a DDE call to a server that did have the required rights? How could the server protect itself from being an unwilling accomplice to a security breach?

Windows provides several API calls that allow a server to impersonate a client. In essence, the calls allow a server to assume the security restrictions of the client in order to determine whether the client has sufficient rights to access a piece of data or a process. For example, a Word-for-Windows user might require access to an Excel data file. The user could gain access to that file using DDE. In this case, the server would need to verify that the Word-for-Windows user has sufficient rights to access the file before it sends the requested data. A server might even find that the client has superior rights when he or she uses this technique. The bottom line is that the server's only concern is for the protection of the data, resources, and environment that it manages.

This set of API calls supports three different types of communication: DDE, named pipes, and RPCs. You need to use a different API call for each communication type. For example, to impersonate a DDE client, you'd use the `DDEImpersonateClient` call. There are some limitations to the level of impersonation support that Windows currently provides. For example, it doesn't currently support TCP/IP connections, so you'd have to resort to using other methods to verify that a user has the proper level of access rights in such a case.

The other security concern is protecting the server itself. What prevents a user who calls Excel from Word for Windows from doing something with Excel that damages the server itself? Ensuring that security concerns are taken care of isn't difficult to do with files and other types of named structures because the file server automatically attaches a security descriptor to these objects. (A DDE server like Excel wouldn't need to do anything, in this case because the file is under the control of the file server.) However, many of the DDE or application server's private objects aren't named and require special protection. Windows also provides API calls to help a server protect itself. For example, the `CreatePrivateObjectSecurity` call allows the server to attach a security descriptor to any of its private objects—say, a thread or other process. The security descriptor would prevent anyone other than the server from accessing the private object.

Understanding How Object-level Security Translates into User-level Displays

By this time, you may have the idea that the concepts that we've been talking about, both security descriptors and access tokens, are so abstract that they'll never see the light of day in the user environment. The opposite is actually true. You'll see evidence of all of the various security elements that we've talked about right in the user environment. In fact, the user will be intimately acquainted with their effects, even if he or she doesn't know the technological names behind what's happening. It's the public nature of these security elements that makes it imperative that you understand how Windows NT security works at both an overview and an in-depth level.

The purpose of this section is to show you examples of both security descriptors and access tokens at work. Although this section isn't designed as a complete reference to the Windows 2000 user interface, we'll cover enough of the user interface that you'll be able to see some of the security features of your application in action as well. The goal of this section is to help you understand how you can use the user interface to test some of the security features of your application. Obviously, there are going to be many security features that you'll need to test in other ways, but this section provides a good introduction.

The first place you'll see security descriptors at work is right in Windows Explorer. All you need to do is right-click any drive icon, choose Sharing from the context menu, and you'll see a display similar to the one shown in Figure 17-5. This folder allows you to create one or more "shares" for the drive.



Figure 17-5: Security descriptors are attached to resources like files, folders, and drives.

Let's take a closer look at how shares work in this case. Each share that you create has a common name and other identifiers. Groups and users are also associated with the share as shown in Figure 17-6. As you can see, this combination of features fits within the descriptions of a security descriptor perfectly. Every share that you create for a drive is an individual security descriptor.

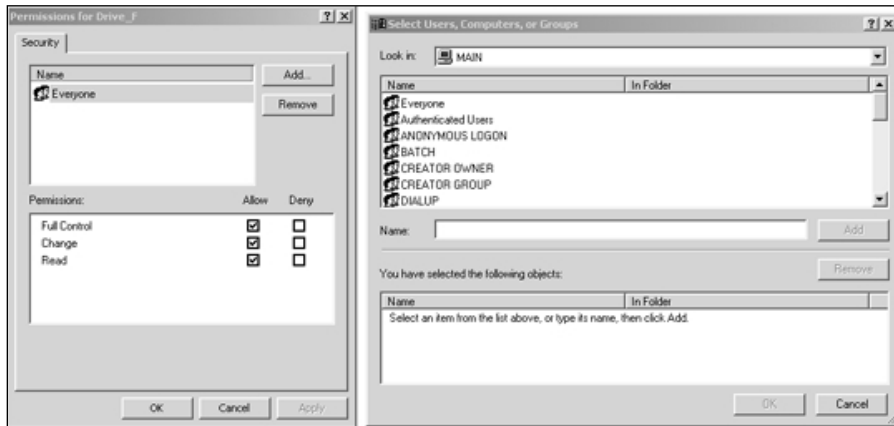


Figure 17-6: Each share is really a security descriptor for a drive from Windows' perspective.

Access tokens are equally apparent within Windows 2000. Remember that an access token is the set of rights that a user has to work with other Windows objects. The user's access token has to match (or exceed) the requirements of the object's security descriptor before the user can access the object.

A user's access token can contain rights to individual objects, or access that's provided by group membership. Giving the user an access token for a single object is simply a matter of assigning the user to the object as part of the object's Security dialog. Figure 17-6 shows an example Security dialog for a disk drive. Granting the user access to all of the objects shared by an entire group means adding a group entry to the user's account. Figure 17-7 shows the Membership tab in the <Username> Properties dialog. This is where you'd add the group membership in most cases.

As you can see, it's possible to see many of the low-level security features that Windows 2000 provides simply by looking in the right places. Though this short trip through the Windows 2000 interface probably didn't provide much from a user perspective, it should have shown how you, as a developer, can use the Windows interface to your advantage. If you're working with security descriptors and access tokens, it's usually possible to test the code in some innocuous way on an interface element first, then use it with the specific object that you want to protect.

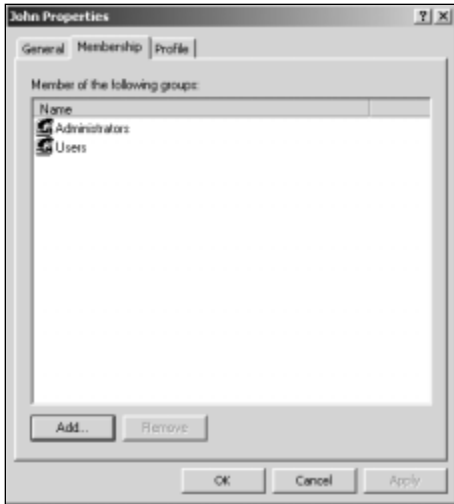


Figure 17-7: Access tokens can be assigned individually, or as part of a group membership as shown here.

An Administrator's View of Security

Just about everyone knows what kinds of problems face the network administrator trying to enforce security on a network today. We all know that short/common passwords, users who don't understand security procedures, and outright neglect on the part of the administrator can all work toward making your LAN a security risk. Fortunately, operating systems like Windows 2000 make it much easier to enforce security on the local machine and the LAN, regardless of user habits. Knowing how to access these security features and build them into your applications is essential to ensuring the safety of both your data and applications.

When it comes to applications, the level of security that Windows 2000 provides is a two-edged sword. On the one hand, there aren't too many things that an application can do to break security. In most cases, Windows 2000 will simply terminate an errant application before any kind of security breach can occur. On the other hand, such stringent security actually breaks some older applications that work fine when using less stringent forms of Windows like Windows 95/98. In essence, the security that Windows 2000 provides can actually affect the level of compatibility your machine can provide. Obviously, if your company intends to create a Windows 2000-only setup, then you need to provide some type of upgrade plan for both custom and off-the-shelf applications to ensure Windows 2000 compatibility. In most cases, you'll need to plan on company-wide application upgrades because the security features of Windows 2000 are just too good to ignore.

Continued

Continued

Sometimes you may opt to get the best of both worlds by ignoring security. If your only intent is to design an application that performs a useful, nonsecurity-related service, you may want to design the application in such a way that it's compatible with all versions of Windows. Unfortunately, there isn't really any middle ground when it comes to security within Windows—at least not from an application perspective. Windows 2000 and Windows 95/98 put two almost completely different faces on the issue of security—Windows 2000 furnishing almost too much security and Windows 95/98 almost none.

Security is also one of the major issues facing anyone building an Internet or intranet site today. It's hard to know who's harder pressed—the company building a public-access Internet or the one trying to hide its presence with an intranet. No matter which environment you work in, a host of companies are preparing tools of various kinds and capabilities to address the needs of those who want to protect their data from harm. Unfortunately, all the nifty tools in the world won't prevent you from shooting yourself in the foot when building your own controls and application programs. (These tools also don't work very well when you use them in the wrong way.) Again, knowing the security features that Windows 2000 provides can help you improve the security of remote-access technologies like ActiveX.

Obviously, when dealing with an Internet site, you have to consider the basics as well as the complicated questions. For example, you need to ask just who is involved with an Internet site and how. It's also important to know about the weak links in your organization and what harm they could cause by downloading controls from the Internet without too much thought. Unless you know whom you're protecting and from what, the effort is a lost cause from the start.

You also have to consider what security problems a specific technology brings with it. Consider the question of Java security: When Sun started creating Java applets, they took what's termed the “sandbox” approach to security. A Java applet can only play in its own sandbox—it can't access the operating system or even the hardware hosting the applet outside the strict areas that Sun thought the applet should access (which is pretty close to no access at all). The runtime engine you need in order to use a Java applet ensures that access is strictly controlled. The advantage of this approach is immediately clear—lack of access also means lack of security concerns. Unfortunately, it also severely limits what you can do with a Java applet, especially when it comes to data exchange of various sorts.

Unfortunately, even Java isn't bulletproof. In the past, crackers have found holes in its implementation that make Java a less-than-perfect solution. For example, on March 22, 1996, Drew Dean (ddean@ICS.Princeton.EDU) and Ed Felton (felten@ICS.Princeton.EDU), of the Princeton Department of Computer Science, found a bug in Java that allowed them to create an applet that deletes a file on the user's local disk. They did this by downloading a file to the user's machine through the Netscape caching mechanism and then fooling Java into running the applet. Fortunately, this bug was fixed in the 2.01 release of Netscape. You can find out about other potential security problems with Java at <http://www.cs.princeton.edu/sip/pub/secure96.html>. In short, crackers find new holes in the various “secure” technologies available to application programmer and administrator alike, just about as fast as new technologies become available.

Microsoft took another tack with ActiveX controls, called the “shrink-wrap” approach. With an ActiveX control, you have full access to the system, just as you would with any shrink-wrapped piece of software. What this means is that you can write some extremely flexible controls that make full use of what the system has to offer. There are no limitations from a data-access standpoint either. However, from a security perspective, ActiveX could turn into a nightmare. What would prevent a control that you downloaded from completely trashing your system as you watched in horror?

There's another aspect to this “shrink-wrap” approach—vendor identification. When you buy a piece of software at the store, you know who produced it. If the software contains a virus or even a nasty bug, you know whom to contact. We'll look at a certification method that Microsoft is proposing for ActiveX controls that does the same thing as the packaging for a piece of software you buy in the store—identifies with certainty the vendor who created the software you're using.

Lest you think that scripts are easily monitored, just take a look at some of the stories in the recent trade press. The low learning curve for scripting languages actually makes them easier to use to circumvent security, not harder. The short version is that neither VBScript nor JavaScript is even close to safe (although more work has been done finding the security holes in JavaScript). The following list presents three of the most common problems:

- ♦ **Tricking the user into uploading a file:** Even though JavaScript has to ask the user's permission to upload a file, a cracker could hide this request in a variety of ways. All a cracker really needs is a button with an interesting caption. Uploaded password files like those used for Windows 95/98 are easily cracked — making the cracker's job of breaking into your system easy.
- ♦ **Obtaining file directories:** A JavaScript doesn't have to ask anyone's permission to upload a directory of your machine. In fact, it can upload the directories of any network machines you have access to as well. A cracker who knows the organization of your hard drive has gone a long way toward being able to break into your system.
- ♦ **Tracking sites visited:** Crackers can learn a lot about you by keeping track of the Web sites you visit. A JavaScript makes this easy to do. It can track every URL you visit and send the addresses to the cracker's machine. As with the file upload problem, the user has to give permission to do this, but the cracker can disguise this permission as just about anything.

Fortunately, there's an ongoing effort to reduce script-related security problems. All the vendors involved in both VBScript and JavaScript are constantly working to improve the security of these two scripting languages. Unfortunately, it's a little difficult to find out what's going on with VBScript. You can read about the currently known security problems with JavaScript at <http://www.osf.org/~loverso/javascript/>.

Kerberos, Public Key Infrastructure, and Distributed Password Authentication Support

Windows 2000 supports three relatively new additions to the tools available to administrators for securing a network: Kerberos, Public Key Infrastructure (PKI), and Distributed Password Authentication Support (DPA). Each of these new technologies is designed to circumvent a particular type of security threat, whether on the Internet or on a local network. Obviously, the developer usually implements anything used by the network administrator. The following list provides an overview of how each of these technologies can help you develop more secure applications.

- ♦ **Kerberos:** This is Microsoft's primary replacement for the Windows NT LAN Manager (NTLM) security currently used to ensure that your data remains safe when using Windows. Kerberos Version 5 is a relatively new industry-standard security protocol devised at MIT that offers superior security support through the use of a private-key architecture. This protocol supports mutual authentication of both client and server, reduces server load when establishing a connection, and allows the client to delegate authentication to the server through the use of proxy mechanisms. Kerberos connects to an online Key Distribution Center (KDC) and the Directory Service (DS) account to obtain session tickets used for authentication purposes.



Windows 2000 will continue to support NTLM when accessing Windows NT 4.0 or earlier. An application can use NTLM for a wide variety of access needs, including remote procedure calls (RPCs), pass-through network authentication, and remote file access.

- ♦ **Public Key Infrastructure (PKI):** This protocol allows two sites to exchange data in an encrypted format without any prior arrangement. The default method for initiating the exchange is to create a secure sockets layer (SSL) connection. The main difference between this technology and others on the market is that it relies on a public-key system of certificates to ensure secure data transfer. The latest specification for SSL is SSL3, which the IETF is calling transport layer security (TLS) protocol. A newer addition to the mix is Private Communication Technology (PCT). PCT still uses public-key encryption, but there are some distinct advantages to using it that we'll discuss later in this section. One of the benefits of using PKI is that there's no online authentication server required because the certificate is issued by a well-known certification authority (normally a company like VeriSign when the technology is used publicly).
- ♦ **Distributed Password Authentication (DPA):** This is a shared-secret authentication method originally started by some of the larger online services like CompuServe and MSN. It allows a user to use the same membership password to access a number of Internet sites when those sites are linked together as a membership organization. In essence, this methodology replicates some of the same features that users can get when using the same password to access multiple servers on a local network. DPA relies on the Microsoft Membership Service for membership authentication and server-specific access information.

All three of these new security methods, along with NTLM, are grouped under the Windows 2000 Security Support Provider Interface (SSPI). Essentially, SSPI allows any application to use any of the security methods to ensure secure transmission of data from one point to another. In fact, SSPI is actually a new Windows 2000 API that you'll be able to use to create more robust applications in the future. SSPI is based on the same architectural concepts as the Internet Engineering Task Force (IETF) Generic Security Services Application Programming Interface (GSS-API). You can find out more about this standard at <http://www.ietf.org/rfc/rfc1964.txt>.

The main purpose of SSPI is to keep the application protocols like DCE RPC separate from the security protocols. Doing so provides an application developer with more flexibility in choosing the kind of transmission and security technology to use in a specific situation.

Hardware-based Security Makes Break-ins Harder

Although software security has gone a long way toward making your PC more secure, some hardware-based solutions are on the horizon that will make the PC more secure than ever. Some of these hardware changes are relatively obvious. For example, Cisco and other router vendors are adding IPSec security as a standard feature to their products. This means that each segment of your network would gain some level of protection from prying eyes. Your direct involvement with a router is going to be limited, but these types of hardware solutions may offer more opportunities to reduce application complexity and make security more automatic.

There are some developer-oriented kinds of hardware security to consider as well. Intel is making changes to the design of the Pentium III that will allow administrators to sleep better at night knowing the machines they administer are at least less susceptible to break-ins. Both of these solutions reside inside the processor itself.

The first processor-specific addition is a unique identification number. This particular feature will make it possible to positively identify the machine being using for a particular purpose. Obviously, this feature could be used to infringe on someone's privacy, so Intel is including the capability to turn this unique number feature off when a machine is used privately. (On corporate machines, only the administrator would have the power to turn this feature off, making it impossible for a user to bypass this security feature.)

The second processor-specific addition is a random number generator that's based on the thermal noise produced by the processor. Software-based random number generators aren't truly random because they rely on complex algorithms and a seed value to produce the number. In other words, the numbers produced by the software pseudo-random number generator will eventually repeat. Because thermal noise is a nonrepeating natural occurrence, the numeric sequence produced by a processor-based random number generator is truly random, meaning that no one can predict what number will be produced in advance.

Continued

Continued

These two features will allow programmers of the future to create nearly unbreakable security for PCs that will definitely keep crackers at bay for quite some time to come. The reliance on hardware to help create these security offerings will keep security monitoring fast. For example, the use of a hardware-based random number generator alone will greatly enhance application speed. The fact that you're getting better output as well only makes these new hardware features more desirable.

By this time, you should have a very good understanding of the various forms of software protection that Windows 2000 provides. However, Windows 2000 goes beyond the offerings made by previous versions of Windows. It also offers support for hardware protection in the form of smart cards. Smart cards are a new credit card-sized device that allows for offline storage of sensitive data of various forms. A complete discussion of this new technology is outside the scope of this book, but it does pay to know that there are other secure technologies available for making your data completely secure.

Caution

A lot of people out there still think it will take weeks to break the 40-bit DES (data encryption standard) security and months to break the 56-bit DES. Nothing could be further from the truth. A recent trade press article noted that a hacking group was able to break the 56-bit security within one day during a RSA yearly DES Challenge contest. The contest was held in San Jose, CA, and the crackers involved (Electronic Frontier Foundation and Distributed.Net) were able to break the previously difficult-to-break DES security within 22 hours and 15 minutes. What this means to you, as a programmer, is that you can't rely on a security scheme to remain secure forever. It's absolutely essential to keep yourself apprised of new security schemes as they become available. You can find out more information about DES Challenge III at <http://www.rsa.com>.

Now that you have some idea of what these new security technologies are all about, it's time to discuss them in a little more detail. The following sections will provide you with a better look at these three new Windows 2000 security technologies and how they can help you create better applications.

Understanding Kerberos

As previously mentioned, Kerberos is a private-key protocol. This means that Kerberos is limited to use between sites that know each other since the keys have to be agreed upon in advance. In other words, you'll normally restrict your use of Kerberos to the local network, a virtual private network (VPN), or intranet communications.



Microsoft is always looking for ways to enhance the security that Windows provides. One of the new features for Windows 2000 is an authentication requirement. A user must be authenticated before he or she gains any access to an NTFS partition at all. In addition, NTFS uses the Kerberos 5 Protocol to secure data on the drive. These upgrades in protection will reduce the probability of someone stealing data from your drive, but doesn't get rid of it altogether. It's always important to combine security monitoring with new security features.

The concept behind Kerberos operation is fairly simple. Kerberos uses the concept of tickets to enforce security. The user must have a ticket for each service that he or she wants to access.

The following sequence of events occurs every time a user requires access to the local server using Kerberos:

1. The user requests access to a server or other resource.
2. The server, in turn, asks for the user's identification.
3. Once the server has gotten the user's identification, it sends the information to the KDC.
4. The KDC checks the user's identity.
5. If the user has the required permissions, the KDC provides tickets for each of the required services or resources.
6. The user provides a ticket for each service or resource needed. This ticket identifies the user and optionally carries the user's access-level information so that the service or resource doesn't provide too much access.

Kerberos represents the primary authentication method for Windows 2000, which is a departure from the NTLM authentication used in the past. The version of Kerberos used in Windows 2000 is fully compatible with the Version 5 Kerberos used for authentication on other platforms. In addition, Microsoft has made it possible to extend Kerberos to allow for nonstandard methods of user identification. For example, you can use smart cards in place of a user name and password for identification purposes.



As previously mentioned, Kerberos is actually an internationally approved authentication standard that is currently being worked on by the Internet Engineering Task Force (IETF). You can find out more about Kerberos at <http://www.ietf.org/rfc/rfc1510.txt>. However, it's also important to mention that Microsoft's version of Kerberos includes some permitted extensions; so though it does provide full standard's based compatibility, it also provides some additional features not found in the standard. The most notable extension is support for smart-card authentication. Another good place to look for non-Microsoft Kerberos information is the MIT site at <http://web.mit.edu/kerberos/www/>. The information on this second Web site isn't as technically intense as the IETF information, but it's a lot more readable.

The KDC will normally run on a domain controller (DC) as a privileged process. Every DC replica will have access to its own KDC to reduce the time required for user authentication. The actual user identification information is contained within the Active Directory, which also runs on the DC as a privileged process. It takes the combination of these two processes to perform user authentication.

The Active Directory also performs other Kerberos-related tasks. For example, it performs multiple master update and replication so that new users can be added at any DC. This same feature allows the administrator to make changes to the user's group membership or change the user's password.

Understanding Public-key Infrastructure (PKI)

We'll discuss two components with regard to PKI. The first is SSL. SSL relies on encryption and digital certificates to do its work. If you're really paranoid about security, you can even combine these two security methods—they're not mutually exclusive.

The second component is PCT. Microsoft and the IETF have worked together to create this new low-level protocol. Like SSL, PCT is designed to prevent crackers from eavesdropping on communications between a client and server through the use of encryption, authentication, and digital signatures. As with SSL, client authentication is optional.

Working with SSL

SSL is a protocol that's fairly easy to understand. Windows NT can (and definitely will) request a digital certificate from the client machine. The client can likewise request a digital certificate from Windows NT. These digital certificates are obtained from a third-party vendor like VeriSign that can vouch for the identity of both parties. However, the certificate exchange procedure is a little more complicated than simply sending a digital certificate from one machine to another.

Here is the six-step process for SSL authentication:

1. The client sends Windows NT an unencrypted random message along with its VeriSign issued certificate (which contains the client's public key). The VeriSign-issued certificate was encrypted using VeriSign's private key. Because everyone has VeriSign's public key, Windows NT can decrypt the certificate and check it for accuracy. Also, because the certificate was encrypted using VeriSign's private key, no one can forge a public key of their own—they have to get it from VeriSign.
2. Once Windows NT confirms it has received a valid certificate and public key from the client, it tells the client to send an encrypted version of its original message.

3. The client computes a digest of its original random message and then encrypts it using its private key.
4. Windows NT uses the client's public key to decrypt the digest.
5. Windows NT compares the decrypted digest to a digest it generates from the random message originally sent in unencrypted form by the client.
6. If the two digests match, the client is authenticated.

At this point, you might ask how a programmer would get involved with what appears to be an administrator issue. Even though Windows 2000 comes with a wealth of built-in methods for controlling access to the server, there isn't anything stopping you from creating an ActiveX component or an ISAPI filter to either augment or even replace the standard security. All you really need to do is monitor the security-related events generated by the server. The most important of these events are the client-authentication requests. Monitoring client-authentication requests allows you to add any requests needed to make sure the party on the other end is actually someone you want wandering around your Web site.

Working with PCT

PCT assumes that you have a reliable transport protocol in place such as TCP. Some people look at TCP/IP as a single protocol, but it isn't. TCP is the transport part of the protocol, whereas IP is the data-transfer portion of the protocol. IP doesn't provide any form of data encryption. So, when you use TCP/IP, your data is open to anyone who wants to see it. Using a protocol such as TCP/PCT or TCP/SSL makes your communications secure. The first version of PCT corrects several problems with SSL, as described in the following list.

- ♦ *Simplified message and record structures:* Reconnected sessions require a single message in each direction if you don't enable client authentication. Even with client authentication, a reconnection requires only two messages in each direction.
- ♦ *Extended cryptographic negotiation:* PCT supports a wider variety of algorithms than SSL. This means it can support a broader range of protocol characteristics and that those characteristics get negotiated individually. For example, the common characteristics include cipher type, server-certificate type, a hash-function type, and a key-exchange type.
- ♦ *Improved message authentication keys:* The message authentication keys are separate from the encryption keys under PCT. This means that messages can use a very long key, ensuring secure transmission even if the encryption key is short or nonexistent. The main reason for this feature is to circumvent the 40-bit key limitation imposed by the U.S. government for secure transmissions.

- ♦ *Patched security hole:* PCT uses a client authentication based on the cipher negotiated during a session. This prevents someone from capturing the client authentication key, disconnecting the original client, and then reconnecting to the server using the stolen key. The client must know both the cipher and the key to gain access to the server.
- ♦ *Addition of Verify Prelude field:* During the original handshaking process, communication between the client and server is carried out in the clear. The addition of this field makes it possible for the client and server to detect any tampering with these “in-the-clear” communications.



Even though SSL version 3 also provides a Verify Prelude field type capability, a cracker can get around it by changing the protocol version number to 2, which didn't include this feature. Because SSL version 3 is fully version-2 compliant, neither client nor server will notice the change.

Microsoft is currently working on the second version of PCT. It's fully compatible with the first version, but offers several important features that the first version didn't. The following list provides an overview of these features.

- ♦ *New datagram record type:* Individual records are sent independently as “datagrams.” In essence, this means that the protocol doesn't guarantee an order of delivery or that the record will even get to its destination. It's up to the client to put the records it receives in order and then verify that they're all present. The main advantage of this approach is speed.
- ♦ *Recognizable record types:* The record header contains information that tells the receiver what kind of record to expect.
- ♦ *Continuation records:* PCT version 1 allowed data to span more than one record even though the record header didn't indicate any form of continuation. Version 2 adds a continuation field to the record header, which allows protocol messages to span more than one record as well.
- ♦ *Intermediate processing of data records:* Data records are encapsulated now, which allows the sender to perform some form of intermediate processing, such as compression.
- ♦ *Independent decryption of datagram records:* Because datagram records could be sent across an unreliable transport, this particular feature is essential to secure communication. Each record is encrypted individually, making it possible for the receiver to decrypt them one at a time even if they're received out of order.
- ♦ *New key management record type:* This record type allows the sender to temporarily change either the encryption or message authentication keys during a session. In essence, this allows PCT to transmit pre-encrypted data.

- ♦ *New closing connection key management message:* This is a special message that tells the other party to close a connection. Because this is an encrypted message, it's harder for a cracker to send a simulated message and close the connection prematurely.
- ♦ *Enhanced message authentication:* Message authentication now includes record headers.
- ♦ *Improved handshaking:* Both the client and server authentication phase include a wider variety of options, including key exchange, signature public key, and certificate.
- ♦ *New private authentication feature:* This feature allows a client and server to authenticate each other using a previously shared identity-associated private key rather than a certified public key.

Now that we have a few of the basics down, let's look at how PCT works. PCT uses variable-length records as a means of communication. Every record contains a header that defines the kind of message it contains. There are two kinds of messages: application and protocol. Application messages always contain data and can use either the standard PCT or datagram formats. A protocol message can key management, error, or handshake information. PCT uses two additional layers. Records are always transmitted using a connection. Normally, there's one connection between a client and server, but there isn't any reason why there can't be more. Every connection is part of a session. Again, normally you'll see only one session between a client and server, but you could have more. (Multiple sessions would require more than one physical connection between the client and server.)

**Web
Resource**

If you want to find out about the current status of PCT, look at <http://www.1ne.com/ericm/pct.html>. This document contains the current draft of the second version of PCT. In addition, you'll want to reference the W3C Web site at <http://www.w3.org/Security/> for details on current Internet security technologies, including PCT and SSL.

A PCT protocol connection begins with a handshake phase. This is where the handshake-management message type comes into play. The client and server exchange several pieces of information, beginning with the negotiation of a session key for the connection. In other words, the client and server decide on a secret password to use for talking to each other. The client and server authenticate each other during this time as well. It doesn't pay for them to talk if the client or server doesn't know who the other party is and whether they can be trusted. Once the client and server determine that they can trust each other, they decide on a master key that's used for encrypting all other messages.

Understanding Distributed Password Authentication (DPA)

DPA is a challenge/response password-based security methodology that relies on a trusted third-party authentication server. It's part of the Microsoft Commercial Internet System (MCIS). MCIS is used as a back-end Internet service that multiple sites can link to, allowing a user to move from one membership site to another without reentering any authentication information (unless required for higher-level access).

When using this protocol, the user provides a user name, which isn't necessarily unique to that user, along with a valid password when requested to do so by a server. The server then requests validation from the authentication server prior to granting access to the user.

Three different levels of user name are supported by DPA: single user, group name, and generic name. The generic name option is the most interesting because it allows the server to access user names like WORD_USERS in place of a specific user names like Tom. In addition, generic names don't have to appear on the server as a specific group as Administrators would.

Windows 2000 Data-encryption Support

Preventing someone from reading your data has been the topic of just about every area of this chapter so far. We've studied protocols and a variety of security techniques that are going to help you make the work of a cracker nearly impossible. However, there isn't a lock that can't be picked given enough time. In fact, that's the whole purpose behind using a 128-bit key instead of a 40-bit key. Picking the 128-bit lock is too time intensive (at least at the moment) to make the data contained in a record attractive to a cracker. It's not that the cracker can't pick the lock; it simply isn't efficient to do so.



A recent trade press article talked about how one vendor's claim of having an unbreakable encryption scheme was causing problems with the cryptographic community. Anyone who constantly deals with security would just love to have a truly unbreakable method of encrypting data. Unfortunately, no matter what the vendor tells you, there aren't any completely unbreakable security schemes for the PC that can protect your data with absolute certainty. The main reason that you can't create absolutely reliable security on a PC is that there isn't any way of generating a truly random number (at least until the Pentium III arrives on the scene in great enough numbers, and developers write code that use its onboard random-number generator). The best you can hope to achieve is a pseudo-random number—one that's generated based on a solid mathematical principle. The

difference between a truly random number and a pseudo-random number is that somewhere along the way the random-number sequence will repeat and someone will be able to predict the next encryption number in a series.

As a result, even given the best math in the world, a dedicated cracker with enough processing power can find ways to get into your system. Even the Data Encryption Standard (DES), which has withstood more than two decades of scrutiny, was recently broken because of the amount of processing power available on the market today (it did take a whole bank of computers to do the job). In sum, always greet any claim of unbreakable security with a healthy dose of skepticism. One of the more interesting Web sites to look at for additional information about data encryption and cryptographic technology is the Counterpane Systems Web site at <http://www.counterpane.com/>.

Layers of Protection

Adding layers of protection is another way to guard against crackers. Encrypting the data at several different levels adds “doors” that the cracker must pass through to get to the data. Put enough doors between your data and the cracker, and the cracker will find something easier to break into.

Microsoft’s CryptoAPI (Cryptography Application Programming Interface) falls into the layer category of protection. It’s a means of adding yet another layer of protection to your sensitive data. Are the encryption techniques you’ll find supplied here unbreakable? No, but they do extend the time required for someone to unlock your data and read it. Using the routines in the CryptoAPI will help you better protect the data transferred between a client and server on the Web.



As with just about everything else dealing with the Internet, the CryptoAPI is new and evolving. At the time of this writing, the CryptoAPI is only mentioned as part of the ActiveX SDK—you’ll need to download a copy of the specification from the Internet. Even though this section provides the best overview possible at the time of writing, you’ll want to visit the CryptoAPI site at <http://www.microsoft.com/security/tech/cryptoapi/default.asp> for the latest information.

The CryptoAPI has another purpose: It’s a general-purpose tool designed to allow encryption of data in any environment, not just the Web. For example, you could build an application that stores data using the same encrypted format whether the information was stored locally, transferred through a modem, uploaded to a Web site, or sent through the mail on a disk. That’s a big advantage to your company. Using encryption all the time for sensitive data means that a cracker breaking into your system will find that the job has suddenly gotten a lot harder, yet a common encryption technique means that your users won’t be inconvenienced by the added security. You can bet that a user is more likely to use encryption when it’s convenient (and for the most part automatic). Making things difficult, hard-to-understand, or simply too time consuming is the best way to convince the user that it’s too much trouble to protect the data on your system.

**Note**

Windows NT 4.0 comes with CryptoAPI 1.0 support built in. When you install Service Pack 3 (or higher), the level of support gets upgraded to CryptoAPI 2.0. This is the same level of support provided by Windows 2000. One of the major differences between the two versions is that CryptoAPI 2.0 provides core cryptographic support, which allows a developer to incorporate cryptography within an application. CryptoAPI 2.0 supports X.509 certificates, ASN.1 encoding, and both PKCS #7 and #10 encapsulation. You'll find the Visual C++ support required to implement CryptoAPI 2.0 support in the WINCRYPT.H and CRYPT32.DLL files.

Microsoft has also moved to a modular approach to their operating design in the past few years, so it's no surprise that the CryptoAPI is modular as well. The CryptoAPI could be compared to the GDI (Graphics Device Interface) API under Windows. Any vendor can add a new device driver that tells the GDI how to work with a particular display adapter. The same holds for the CryptoAPI. It uses the idea of a Cryptographic Service Provider (CSP) just as you would a display adapter device driver. Windows will come with one CSP—the one provided by Microsoft. However, if Microsoft's encryption feature set doesn't meet your needs, you can either design a new CSP yourself or buy one from a third party. The installation procedure will work much like the one you currently use to add a device driver. In fact, this device-driver approach makes it easy for you to mix encryption hardware and software on your machine—a real plus in a world where you normally install hardware and software as separate entities.

**Web Resource**

Microsoft has developed a DDK (Device driver Development Kit) for the CryptoAPI called the Cryptographic Service Provider Developer's Kit (CSPDK). You can get more information about it at <http://www.microsoft.com/workshop/security/capi/cryptapi.asp>.

Encrypting a File

One of the easiest ways to show you the functionality provided by the CryptoAPI is to give an example. Encrypting a file is fairly simple. Microsoft follows a straightforward procedure using an eight-step process (six of the steps involve cryptographic-specific calls). You begin by opening a source and destination file, just as you would in any program. (The example code in this section is written in C, but you can easily port the required header files to other languages.) Once you have valid file handles, you need to get the handle for a CSP, like this:

```
// Get handle to the default provider.
if(!CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_FULL, 0))
{
    MessageBox("Error during CryptAcquireContext", NULL, MB_OK |
    MB_ICONEXCLAMATION);
    PostQuitMessage(1);
}
```

**Note**

You'll find a complete list of cryptography-related functions, structures, and defines in the WINCRYPT.H file.

`CryptAcquireContext()` can accept up to five parameters, but only two of them are essential in this case. The first parameter stores a handle for the CSP. The second parameter specifies the type of CSP that you're looking for. Every CSP has both a name and a type. For example, the name of the CSP currently shipped with Windows is "Microsoft Base Cryptographic Provider v1.0," and its type is `PROV_RSA_FULL`. The name of each provider is unique; the provider type isn't. The second parameter contains the key container name. If you specify a value here, Windows will look for a specific key container. A vendor can store the key container in hardware, within the registry, or on the hard drive, so you normally won't know the name of the key container. Using `NULL` (as shown) tells Windows to return the default key container. The third parameter contains the name of a CSP. You can retrieve the values of any installed CSPs using the `CryptGetProvParam()` function. Supplying a `NULL` value returns the default CSP. The final parameter contains one or more flags. Normally, you'll set this value to 0 unless the CSP provides specific flag values for you to use. (Microsoft does provide some default flag values, but they're for administrative purposes only.)

The next step is either to encrypt the file using a random key or to generate a key using a password. The random-key method is the one you'll probably use most often within an ActiveX control, so that's the one described here. The password method is similar and actually requires fewer steps than the random-key method.

```
// Create a random session key.
if(!CryptGenKey(hProv, ENCRYPT_ALGORITHM, CRYPT_EXPORTABLE,
&hKey))
{
    MessageBox("Error during CryptGenKey", NULL, MB_OK |
MB_ICONEXCLAMATION);
    PostQuitMessage(1);
}
```

The `CryptGenKey()` function provides you with a unique key. Notice that the very first parameter is the CSP handle that we obtained in the previous step. The second parameter contains the name of an encryption algorithm. Microsoft supports two: `CALG_RC2` (block cipher) and `CALG_RC4` (stream cipher). You'll probably find that the algorithm names vary by vendors—so it's important to know which vendor the user intends to use as a CSP. The third parameter contains flags. There are three default-flag values. `CRYPT_EXPORTABLE` tells Windows that it can export the random-key value into a blob (we'll look at blobs in a few moments). `CRYPT_CREATE_SALT` tells Windows to use something other than 0 for the random-key seed value. Finally, `CRYPT_USER_PROTECTED` tells Windows to notify the user when certain actions take place while using this key. Each CSP will probably define a unique set of actions. The last parameter is a container for the returned random key.

Sometime during the process of getting a certificate, the user received a public key. This is the key that the CSP keeps stored in a central place—it identifies the user. To make data transfers absolutely safe, then, it's important to get a copy of the user's public key to use in the encryption process. That's what we do in the next step.

```
// Get handle to key exchange public key.
if(!CryptGetUserKey(hProv, AT_KEYEXCHANGE, &hXchgKey))
{
    MessageBox("Error during CryptGetUserKey", NULL, MB_OK |
MB_ICONEXCLAMATION);
    PostQuitMessage(1);
}
```

The very first parameter we supply to `CryptGetUserKey()` is the handle to the CSP. The next parameter tells what kind of key to retrieve from that provider. Every CSP will support two keys: `AT_KEYEXCHANGE` (exchange the public key provided for a personal certificate used in an application like a browser) or `AT_SIGNATURE` (get a signature key like the one we'll generate later for an ActiveX control). The third parameter provides a place to store the key returned by the function.

Now comes the fun part. We take the random key and the user's public key and mix them together. Mixing them together means that even if the user's key has been compromised in some way, the data will still get encrypted in a way that forces the cracker to start from scratch every time. It's the random element that makes this encryption technology at least a little more secure than other methods.

```
// Determine size of the key blob and allocate memory.
if(!CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, NULL,
&dwKeyBlobLen))
{
    MessageBox("Error computing blob size.", NULL, MB_OK |
MB_ICONEXCLAMATION);
    PostQuitMessage(1);
}
if((pbKeyBlob = malloc(dwKeyBlobLen)) == NULL)
{
    MessageBox("Error allocating memory for blob.", NULL, MB_OK
| MB_ICONEXCLAMATION);
    PostQuitMessage(1);
}

// Export session key into a simple key blob.
if(!CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, pbKeyBlob,
&dwKeyBlobLen))
{
    MessageBox("Error during CryptExportKey", NULL, MB_OK |
MB_ICONEXCLAMATION);
    PostQuitMessage(1);
}
```

Notice that the process of exporting the blob key (the combination of the user's public key and the random key) is actually a three-step process. We call the `CryptExportKey()` function twice. The first two values are the keys we created: random and user. The third parameter tells Windows what kind of blob to create. Most CSPs support two values: `SIMPLEBLOB` or `PUBLICKEYBLOB`. The fourth parameter is for flags. Always set this value to 0 unless a CSP uses it for some purpose. The fifth parameter points to a buffer used to store the blob. If you call `CryptExportKey()` with this parameter set to `NULL`, it simply returns the size of the buffer you need in the sixth parameter. That's what we've done in the first call to the function. The sixth parameter normally contains the size of the buffer when you actually call `CryptExportKey()` to create the blob. (The source code displayed here also contains a memory allocation function call.)

Now that we have a blob to work with, it's time to do a little maintenance. One of the things you'll want to do without waiting too long is destroy the user's public key. Just think about a cracker looking around in memory for some way of breaking down your code. Leaving the key in memory is one of the fastest ways to do this. Killing the key, then, isn't only a good idea from the standpoint of memory and good coding practice, it's a necessary part of maintaining a secure environment. The following source code calls the `CryptDestroyKey()` function to get rid of the public key.

```
// Release key exchange key handle.
CryptDestroyKey(hXchgKey);
hXchgKey = 0;
```

At this point, you'd send the blob data to the destination file on disk. The blob actually forms a header that the receiving computer will use later to decrypt the file. However, a header isn't much good without some data, so that's what we'll take care of next. The first thing you'll want to do is define a block size for the data. If you're using a block cipher, the CSP will provide a block size for you. A block cipher normally requires you to add one additional blank block at the end of the file as well. When using a stream cipher, you can define any convenient block encryption size (although Microsoft recommends a block size of 1,000 bytes). Now you'll read the data into a buffer one block at a time. Use the following code to encrypt it, and then write the block to the destination file.

```
// Encrypt data.
if(!CryptEncrypt(hKey, 0, eof, 0, pbBuffer, &dwCount,
dwBufferLen))
{
    MessageBox("Error during CryptEncrypt", NULL, MB_OK |
MB_ICONEXCLAMATION);
    PostQuitMessage(1);
}
```

You can process the data in a loop until you run out of blocks. Make sure you pad the end of any incomplete blocks. The CryptoAPI depends on an even block size as part of the decryption process.

Internet Security Issues

The overabundance of security options that are provided by Windows 2000 is a big plus for both Internet users and sites. It's the level of native security that makes Windows 2000 an excellent platform for a Web site. In addition, you can improve on most of the standard features provided by a Web site if you go to the added trouble of writing code to use the features that Windows 2000 provides. For example, we saw in Chapter 9 that you can write ISAPI filters to monitor events in the background. Combining an ISAPI filter with the security features of Windows 2000 could provide everything you need to at least track significant security events. In other words, if you can't prevent a cracker from breaking your security, you can at least track what they're doing to keep damage to a minimum. In addition, knowing where someone did break in is one way to improve security in the future.

Security features need not be limited to ISAPI filters or desktop applications; you can also incorporate Windows 2000 security into an ActiveX control or an ISAPI extension. For example, you can write an ActiveX control that tests whether a person logging into the system is actually allowed access to the server. Likewise, an ISAPI extension could display a login screen for the user and then check for access level in a system database. You can even write an ISAPI extension that would send different kinds of HTML pages back to the client based on the level of access the user is allowed by the system.

Theoretically, you can also add Windows 2000 security into DLLs designed to be accessed from a client machine (even a Windows 95/98 client) using DCOM. Because the DLL will execute on the Windows 2000 server rather than being downloaded to the client (as is normally done), the client could be protected using standard Windows 2000 security features. Realistically, adding this kind of security to an application would prove expensive when it comes to network bandwidth and server processor cycles. You'd want to implement security through DCOM in the minority, rather than the majority, of cases.

So, why would you want to go to all the effort of building additional Windows 2000-specific security into your application? For one thing, there are holes in the current API specifications for Internet security. There are methods that an unscrupulous person could use to bypass all of the security features currently provided for Internet-specific situations as a result of holes in the current defenses provided by the various browsers on the market.



Part of your protection strategy is going to include testing multiple browsers to see how they react—especially if you're creating a public-access Web site. As a programmer working with multiple products, you still have to keep track of all the Web sites you visit on a regular basis for ideas. A product called NavEx allows you to create copies of your Internet Explorer Favorite Places folder as Netscape bookmarks and vice versa. You can download it at <http://mach5.ocs.drexel.edu/navex/>.

These security holes aren't in the API itself, but in some of the creative solutions people used in the past to make the Internet work. (We'll concentrate on problems found in various versions of Internet Explorer, but you can be sure Netscape and other browsers have the same, slightly different, or even worse problems.) In essence, problem solving in the past created security back doors in the present. Table 17-2 tells you about three of the holes in the Internet Component Download service (although it's almost certain that more holes exist). For the most part, you can count on these holes being fixed by the time you read this chapter; the problems are listed as examples of things you need to be worried about more than to provide an actual list of existing problems.

Table 17-2
Examples of Holes in the Internet Component Download Service

<i>Hole Type</i>	<i>Description</i>
HTML <A HREF> tag	There are several ways to download and run an EXE file using the <A HREF> tag. The current method used by Internet Explorer 3.x/4.x to keep this problem in check is to have the HTML parser use the URL moniker directly to download the code and then to call WinVerifyTrust to check code validity. Is this method 100 percent safe? No, because you're using something other than the standard procedure to verify the contents of a file. In this case, you're relying on the HTML parser.
Scripts	Right now scripts are totally free of any kind of security check. There's no way to verify who created the script or what it might do to your machine. More importantly, there isn't any way to verify what information the script might retrieve from your machine. (We looked at two kinds of information a JavaScript could retrieve from your machine earlier in this chapter: the directories on the user's hard drive and the sites the user visits.) Microsoft is working to create some kind of script certificate. Once script certificates become a reality, the browser can call WinVerifyTrust to check a script before running it.
Full applications or other complex download situations	Internet Explorer does a good job of checking specific kinds of downloads right now. For example, downloading an OCX initiates a WinVerifyTrust sequence. What happens if the download parameters fall outside the limited scope of things checked by Internet Explorer? For example, a user might want to download and install Doom or some other game program. The installation sequence might include unpredictable actions such as making registry entries and rebooting the machine. Internet Explorer can't handle that situation at the moment. Microsoft plans to make future versions of component download more robust so that it can handle such events.

One of the first steps in understanding how Windows 2000 can help you enhance the security of your Internet site is to look at the basic security features offered by the operating system itself. Every security feature like drive mapping that Windows 2000 has to offer can be accessed using an application. What many programmers seem to forget is that you can also access those features using an ActiveX control. For example, you can create a button similar to the one we created in Chapter 12 whose whole purpose is to display a Map Network Drive dialog in which the user can choose a drive to map on the network.

Of course, some Windows 2000 security features, such as mapping a drive, don't make much sense in the realm of Internet security—creating a drive mapping pushbutton would make sense only if you intended to use it in an application. However, adding other Windows 2000 security features to an ActiveX control does make sense. For example, you can use the password-protection features of Windows 2000 to display a logon dialog every time someone requests secure access to your site.

Creating an ActiveX control to display a logon dialog actually accomplishes two things in an Internet application. First, the password security provided by Windows 2000 is a lot better than that provided by Windows 95/98. A cracker will have a pretty tough time getting past the logon dialog without the right password. Second, you can tell the server to log every secure access—remember that Windows 2000 gives you the capability of monitoring everything. If someone does manage to break into your system, you'll at least know which account was used. Having an account name will allow you to assess the level of damage the cracker could inflict based on the security level of the person whose account was broken into.



Sometimes you need more than software security for your system; you need physical security as well. For example, you might want to provide a disk drive on most user workstations to make them easy to update and maintain, but not want the user to access the drive to add unauthorized and potentially virus-laden software. One way to ensure that no one can access the system through the disk drives is to place a physical lock on the drives like the one provided by the Safety Claw. This device uses a side lock on the cabinet. An angled bar goes through the side lock and blocks the front of the disk drive, making it impossible to open. You can find out more about the Safety Claw and other physical security devices at the INTERFACE Security Solutions Web site at <http://www.crocodile.de>. This Web site provides both English and German pages for devices that can protect just about any part of your PC (both laptop and desktop machines). You'll also find physical security devices for printers, mice, and scanners.

Understanding DCOM Security

DCOM allows a client to request the launch of a component on a remote machine, then access the services that the component provides through the remote connection. This means that DCOM has the potential for introducing unexpected or unanticipated security problems because the component could actually give the user access to information that he or she normally couldn't access. In addition,

DCOM introduces the possibility of a client remotely launching a harmful application that could have a detrimental effect on the operation of the server. In other words, a developer has to exercise greater caution with a DCOM application than with other application types, including client/server where the operation of the server-side application is strictly under the control of the server itself.

Unlike many other parts of Windows, DCOM has to take other computing platform requirements into account. With this in mind, DCOM supports a replaceable security model that uses whatever security model the target platform can support. For example, if you write a component that resides on a Windows NT 4 machine and try to access the component from a Windows 2000 machine, DCOM itself will more than likely use NTLM as the security protocol. On the other hand, if all of the machines involved in a DCOM application used Windows 2000, then DCOM would rely on Kerberos for security. DCOM exposes its security model through the SSPI API that we discussed earlier in the chapter. Obviously, the ability of DCOM to use replaceable security becomes more important as you consider how DCOM will interface with other platforms like Unix.

You can set up DCOM security in two relatively easy ways. The most common method is to use the DCOM Configuration Tool that we discussed in Chapter 13. This is the easier of the two methods and is probably the method you should rely on when working with DCOM security. An alternative to using the DCOM Configuration Tool is to work with the OLE/COM Object Viewer utility in expert mode. Because this particular method gives you a lot of information about the component in question, you'll find that you can better understand how the component is installed. However, this method also requires a better knowledge of how COM affects the registry — truly making it the choice of the expert user.

Sometimes it's easier to understand how the security features of a technology work if you break them into pieces. DCOM provides secure communications for both the client and server in four ways. The following list gives you a brief outline of the four ways:

- ♦ **Access Security:** This security level answers the question of who is allowed to access the component. It determines who is allowed to call upon the component for the services that it provides.
- ♦ **Launch Security:** This security level determines who can create a new object within an existing process. In other words, this level looks at who can create threads within the component once it's launched.
- ♦ **Identity:** This security level determines the capabilities of the component itself from a security standpoint. For example, it determines what area of the hard drive the component will be able to access.
- ♦ **Connection Policy:** This level of security determines which type of message security gets implemented. For example, it determines whether messages can be altered. It also looks at things like the originator of the message. Can the server even determine who the originator of the message is and what level of access this individual has?

Now that you've had a chance to look at the big picture, let's talk about the various levels of security in more detail. The following text will tell you more about the four ways in which DCOM ensures that your data will remain secure.

Access Security

There are a number of issues when it comes to access security. Your first inclination might be to lock the component up so that no one can access it without authorization. Of course, this would never work with an Internet connection where a user might log on as anonymous and still want to access the database of parts that your company wants to sell. In other words, sometimes you have to create a broader-based access security scheme. Such a scheme might severely limit component method access for people who log on as anonymous, while providing better access to those who provide a user name and valid password.

DCOM currently provides declarative access on a per-process level. You can provide varying levels of access security by simply changing the security of the instance of the component itself.

Obviously, trying to determine who is calling on the component and then modifying the security level of the object instance as appropriate isn't always easy or even possible. In some cases, you might have to add some form of security to the component itself. An application could be designed in such a way that it would cooperate with the component and ask the user to provide additional information as needed.

Launch Security

One of the strengths of COM is that it does indeed provide a fully functional component environment. The DCOM component that a client launches has full access to all the other components on the server. This represents a security risk since a rogue component could theoretically circumvent security. A client could launch the rogue component, which would then gain access to components that the client normally might not be entitled to launch.

Launch security ensures that a component doesn't gain access to other components that it isn't authorized to use. Normally, you'll ensure that launch security is maintained through three methods. First, give each object its own security profile as described in the "Identity" section that follows. Second, maintain control over the registry by making registry entries secure. Using secure registry entries ensures that a client can't even find out about a component that it's not authorized to use. Third, add internalized component security checking. Make sure that your component actually checks the identity of the client that's trying to launch it before allowing the client to do so.

Identity

Objects require some form of individual security identity if you want to maintain the safest possible environment for your server. Ensuring the client has the right to create the object doesn't necessarily mean that the object is free from security constraints. It's always a good idea to ensure that components are kept from corrupting the server security environment as well by having a security identity of their own. There are two main methods for enforcing component security: user level and object specific.

In some cases, you can simply force the launched component to assume the same security profile as the client who launched it. This ensures that the component doesn't have any more access to the server than the client does. Unfortunately, there are a couple of drawbacks to this approach. First, to use this method you have to limit a single instance of an object to use by a single user, which isn't always possible or efficient. Second, the component may legitimately require access to other parts of the machine that you don't want the client to access directly.

Another way to ensure launch security is to assign the component its own set of privileges that aren't reliant on the client. This approach will limit the amount of risk to the server, yet allow the component to perform all of the tasks that it's designed to do. Of course, if you go this route, you'll need to exercise caution when setting the component's rights. Give the component too many rights and you've exposed the server to undo security risk; too few rights may prevent the component from working at all.

Connection Policy

The "wire" between the client and server isn't necessarily secure when you use DCOM. There's no way to tell if the connection has occurred within the same building, between buildings within a secure compound, through a secure intranet connection, or over the completely open connections of the Internet. As a result, it's absolutely essential to maintain some form of connection policy — a security measure that tries to determine if the messages between the client and the server have been modified in some way.

One of the problems that you'll face when trying to enforce a connection policy is the source of the message corruption. For example, a message change doesn't necessarily have to originate with a cracker intent on learning your company's secrets; it could simply be a matter of a physical connection problem on the network or noise introduced by man-made or natural sources. Fortunately, network hardware has progressed to the point that it will automatically correct most forms of message corruption automatically, but it's still important to realize that this form of message corruption can occur.

My discussion in this section is based on the assumption that the threat you're most interested in protecting against is the cracker who wants to get into your company's network. DCOM does provide two levels of data protection: integrity and privacy. The following list describes each type:

- ♦ **Integrity:** Ensures that the data gets transferred from the client to the server without being modified. However, this level doesn't stop a third party from intercepting the message, reading it, and passing it along unchanged.
- ♦ **Privacy:** This level provides both data encryption and data-integrity checks. In other words, DCOM will encrypt the data so that it can't be read by third parties and will also check to make sure that no one has altered the message.

Using the Security Configuration Editor

The Microsoft Security Configuration Editor is a new MMC snap-in designed to make life a lot easier for the network administrator by reducing both security management and security analysis time. Initially, you'll use this tool to configure the various operating system security parameters. Once these parameters are in place, you can use the Security Configuration Editor to schedule periodic tests. These automatically executed tests help ensure that your operating system maintains the level of security that you initially set up for it.



Note

You can use the System Configuration Manager MMC snap-in to work with the security database (SDB) and security configuration (INF) files you create using the Security Configuration Editor.

The overall goal of the Security Configuration Editor is to provide a single place to manage all of the security concerns for a network. However, it doesn't actually replace all of the tools you used in the past — the Security Configuration Editor augments tools like the User Manager, Server Manager, and the Access Control List (ACL) Editor. The Security Configuration Editor is also designed to provide auditing tools that Windows has lacked in the past. When working with Windows NT 4, the developer had only the Event Viewer to use, which obviously doesn't provide much in the way of information.

One of the unique ideas behind the Security Configuration Editor is that it's a macro-based tool. You must create a set of instructions for the Security Configuration Editor to perform, then allow it to perform those instructions in the background. Obviously, this saves a lot of developer time since the developer doesn't have to wait for one set of instructions to complete before going to the next set. Tasks can also be grouped, which further saves input time because all of the instructions required to implement a given setup can be configured at one time.

At this point, you may wonder why a developer should care about this tool at all. After all, configuring network security is a network administrator's task. That idea

used to be true—a network administrator was responsible for all security on the network. However, as computer networks have become more complex and the technologies used with them more flexible, part of the responsibility for network security has shifted from the network administrator to the developer responsible for creating the various components that make the network operational. As a developer, you need to know how this tool works so that you can test the applications you create.

Creating a security setup begins when you either choose an existing template or create a new one. If you want to use an existing template as a basis for creating a new one, you can right-click the desired template and use the Save As command found on the context menu. Microsoft supplies a variety of templates designed to get you started in creating this security database, as shown in Figure 17-8.

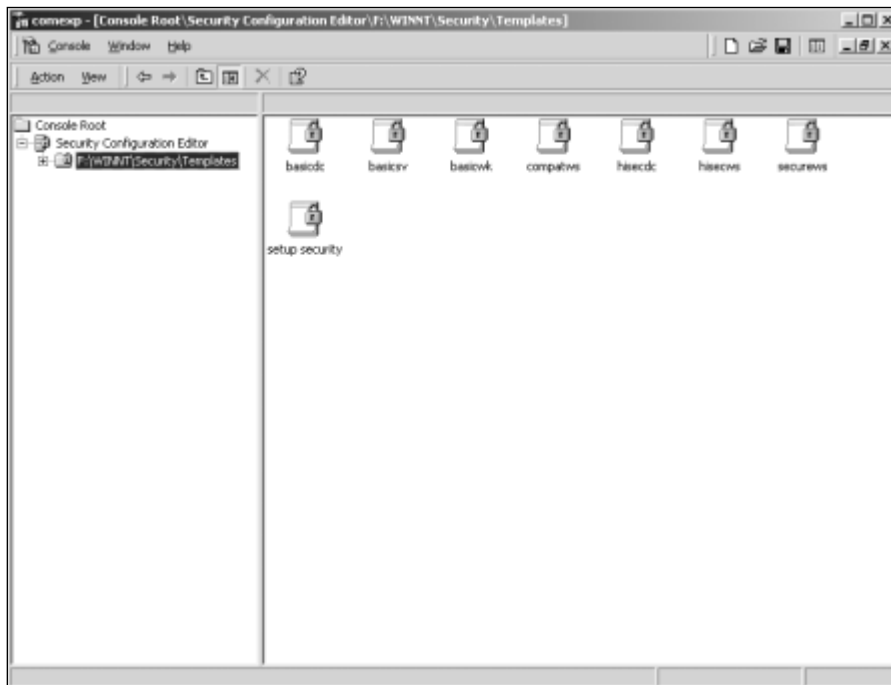


Figure 17-8: The Security Configuration Editor provides you with a number of standard templates to use when creating your security setup.

Each of the security templates is designed for a different purpose (which is indicated by the name). The one I'll be using throughout this section is the basic workstation template (basicwk), but all the other templates will work about the same as this one. All the templates contain the same basic elements shown in Figure 17-9. The following list describes each of these elements for you:

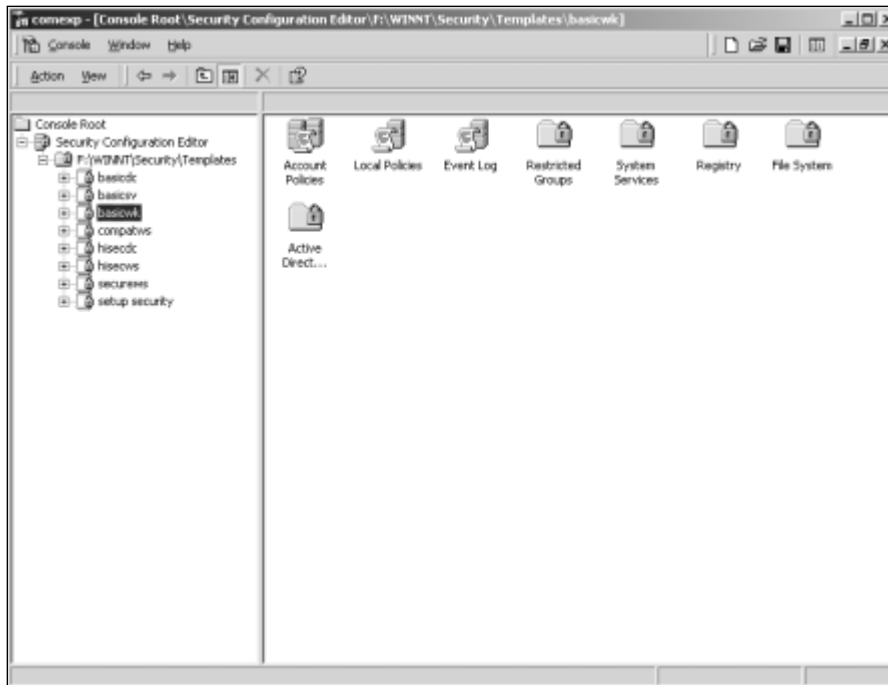


Figure 17-9: Each of the security templates contains the same security elements.

- ♦ **Account Policies:** Defines the password, account lockout, and Kerberos policies for the machine. Password policies include items like the minimum password length and the maximum time the user is allowed to use a single password. The account lockout policy includes policies like the number of times a user can enter the wrong password without getting locked out of the system. Kerberos policies feature elements like the maximum user ticket lifetime.
- ♦ **Local Policies:** Defines the audit policy, user-rights assignment, and security options. Audit policies determine the types of data you want to collect. For example, you could choose to audit each failed user logon attempt. User-rights assignments should be of special interest since this policy affects the kinds of rights you can assign to a user (the access token). The security options policy contains all of the elements that determine how the security system will react given a specific set of circumstances. For example, there's a policy that will forcibly log a user off when his or her usage hours expire.
- ♦ **Event Log:** Defines how the event log stores data and for how long. This set of policies also determines event log size, which can view various event logs, and what to do if the event log gets full.

- ♦ **Restricted Groups:** Defines groups that aren't allowed to access the workstation or server at all, or restricts the amount of access they can obtain.
- ♦ **System Services:** Displays a complete list of all of the system services on the target machine. Double-clicking a service will display a dialog that allows you to set the security policy for that specific service. This same dialog will allow you to adjust the startup mode for the service. Normally, you'll want to leave the standard icons in this policy alone. However, you can safely change any system-service DLLs you create.
- ♦ **Registry:** Contains all of the major branches of the registry. Double-clicking a branch displays a dialog that allows you to set the security level for that branch. In addition, you can choose the method of security inheritance by children of this branch.
- ♦ **File System:** Contains a variety of protected-file system entries. You can add new files to the list or modify existing entries. Double-clicking a file system entry displays a dialog that allows you to set the security level for that file system member (which could be a folder or a specific file). In addition, you can choose the method of security inheritance by children of this file system entity (applies only to folders).
- ♦ **Active Directory Objects:** This entry is available only if you have Active Directory enabled (which means you must have a domain controller set up). It allows you to edit the security settings for any Active Directory objects, including users and groups.

Once you define a security template, you simply right-click the template and choose the appropriate option to apply the template's policies to the system. Optionally, Microsoft plans to provide a command-line utility that you can use to apply the template, or you can use the Security Configuration Manager snap-in to apply the template. Of all the options for applying a template, the Security Configuration Manager snap-in for MMC is probably the best. The following steps will get you started applying a template to a specific machine.

Try This

1. Right-click the Security Configuration Manager icon and choose Open Database from the context menu. You'll see an Open Database dialog like the one shown in Figure 17-10.

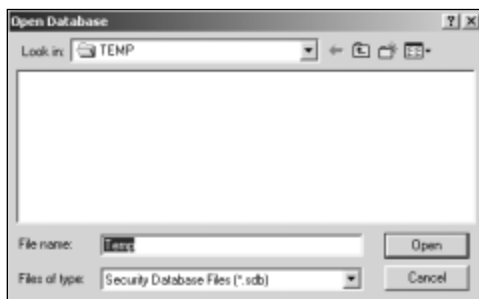


Figure 17-10: The Open Database dialog allows you to open an existing security database or create a new one.

2. Type the name of the new database you want to create, and then click Open. You'll see a Select Configuration to Import dialog like the one shown in Figure 17-11. This is where you need to enter the name of the template that you've created using the Security Configuration Editor.

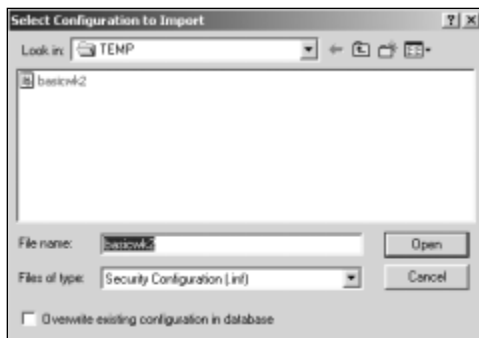


Figure 17-11: Make sure you know the location of the security template you want to apply to the current system.

3. Find the template file on your drive, highlight it, then click Open. At this point, the Security Configuration Manager will start analyzing the template. You should see a Perform Analysis dialog like the one shown in Figure 17-12. This dialog allows you to choose a location for the error log that the analysis process will create. It's important to view the contents of this log later so that you can fix any problems with your template before attempting a failed application again.

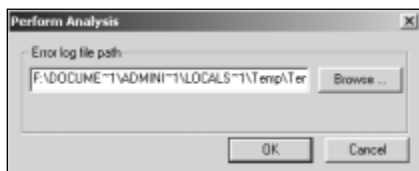


Figure 17-12: The Perform Analysis dialog allows you to choose a location for the error log.

4. Choose a location for the error log, and then click OK. Security Configuration Manager will display the Analyzing System Security dialog shown in Figure 17-13 as it performs the analysis of the security template file.

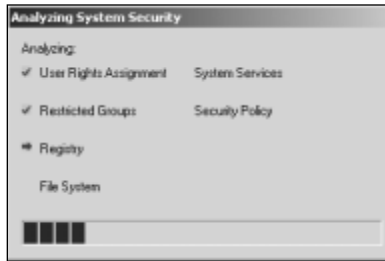


Figure 17-13: The Analyzing System Security dialog shows you the progress of the analysis process.

5. After the Security Configuration Manager is finished, you'll see the standard Security Configuration Editor display shown in Figure 17-9. However, in this case, you won't see any of the template names. These are the security settings that you've just added to a security database.
6. Right-click the Security Configuration Manager icon, and then choose Configure System Now. You'll see the warning dialog shown in Figure 17-14. Click OK and the System Configuration Manager will apply the new policy to your machine.

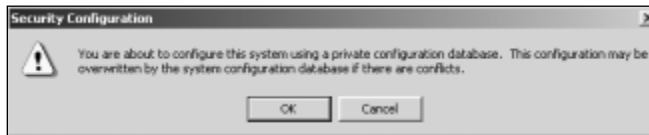


Figure 17-14: The System Configuration Manager will display this warning before it applies the new security policy to your machine.

At this point, you have enough information to work with the Security Configuration Manager. As you add new components to the system, it's important that you test various security options for them, then work with the network administrator to create a policy for the component.

Summary

Understanding how security works is an important part of learning what you can expect from your security system. Every form of security available today has some kind of potential flaw, some place where a cracker can get in and damage your data. However, the form of security that you choose can at least deter novice crackers who don't have the experience to handle complex systems and delay entry by

others. In addition, a good security system will tell you when it has been breached so that you can take the appropriate countermeasures. Here are some additional conclusions you can draw from the contents of this chapter:

- ♦ Ensuring the interoperability of the various security solutions you implement is an important part of application planning.
- ♦ Hardware-based security features that you'll find in tomorrow's processors will definitely ease the problem of ensuring your data remains safe. For example, Intel's Pentium III processor will include both a unique identification feature and true random-number generator, both of which will give programmers the tools they need to make software safer.
- ♦ Never assume that any software solution will enable you to provide complete and totally reliable security for your network; always assume that a cracker can break into your system given enough time and computing power.
- ♦ Security descriptors determine what level of security is attached to an object and what a user needs to access the object, whereas access tokens determine what rights that user has. If the user's access token has a level of security equal to or greater than the requirements defined by the security descriptor, then the user can use the object.
- ♦ The SACL works with Windows' auditing feature, whereas the DACL controls who can actually use an object.
- ♦ An ACL consists of two components: a header containing the number of ACEs for the ACL, and the ACEs themselves.
- ♦ Windows 2000 supports four main types of security: NTLM, Kerberos, PKI, and DPA. Kerberos is the main security provider, whereas NTLM is supplied for compatibility purposes with older versions of Windows.
- ♦ Microsoft's CryptoAPI enables you to protect the data sent from one site to another using data encryption.
- ♦ Physical security is often a required add-on for systems that are exposed to possible threats from within an organization. For example, if you can't lock a server away from public view, the next best thing is to physically secure the server so that no one can access it without a key.
- ♦ Either the DCOM Configuration Tool or the OLE/COM Object Viewer utility will allow you to set up DCOM security. However, the DCOM Configuration Tool is the utility of choice for most people because it's easier to use.
- ♦ You need to work with the four levels of DCOM security to create a secure environment: access security, launch security, identity, and connection policy.
- ♦ A connection policy can be enforced at two levels: privacy and integrity. The privacy level provides encryption and data-integrity checks.
- ♦ You'll use the System Configuration Editor to create security templates for a machine and the System Configuration Manager to actually deploy those security policies.

